# Machine Learning 1

Hayoung A15531571

10/21/2021

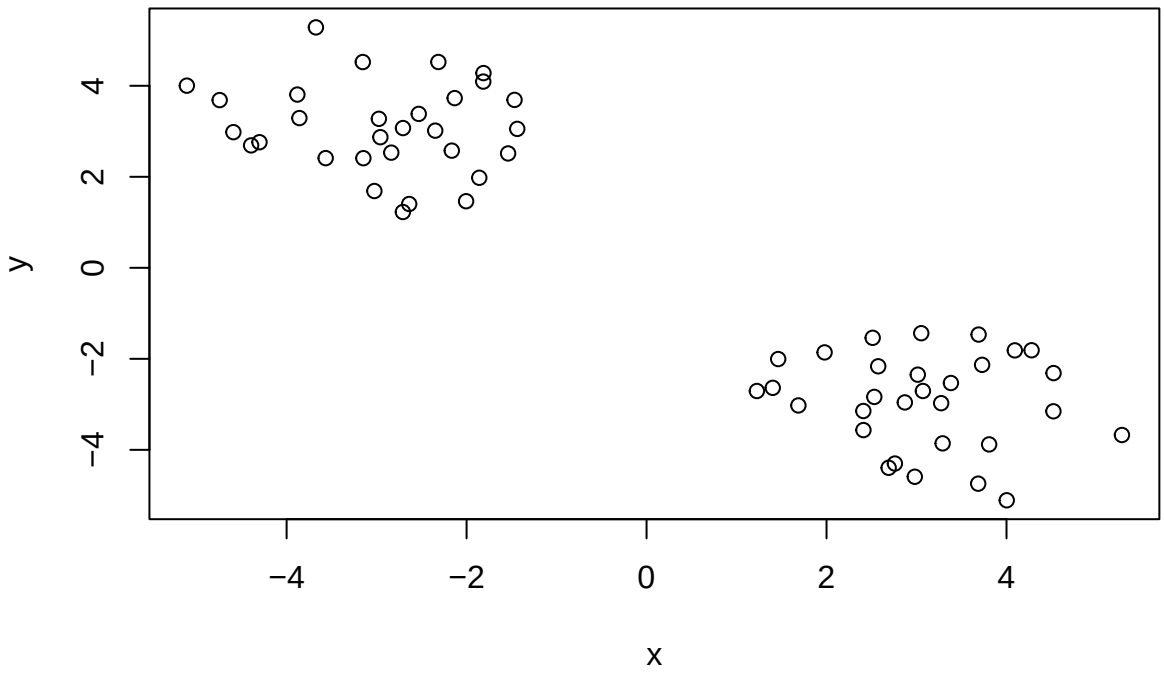
First up is clustering methods

#Kmeans clustering

The function in base R to do Kmeans clustering is called `kmeans()`

Generate some example data for clustering where we know what the answer should be rnorm() gives a random set of normalized data. In this case 30 values centered around -3 and 3

```
tmp <- c(rnorm(30,-3), rnorm(30,3))
x <-cbind(x=tmp, y=rev(tmp))
plot(x)
```




Q. Can we use kmeans() to cluster this data? setting k 2 and nstart to 20?

```
km <- kmeans(x, centers = 2, nstart = 20)
km
```

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1  3.073150 -2.922852
## 2 -2.922852  3.073150
##
## Clustering vector:
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 59.0435 59.0435
##  (between_SS / total_SS =  90.1 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How many points are in each cluster?

```
km$size
```

```
## [1] 30 30
```

Q. What 'component' of your result object details cluster assignment/membership?

```
km$cluster
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

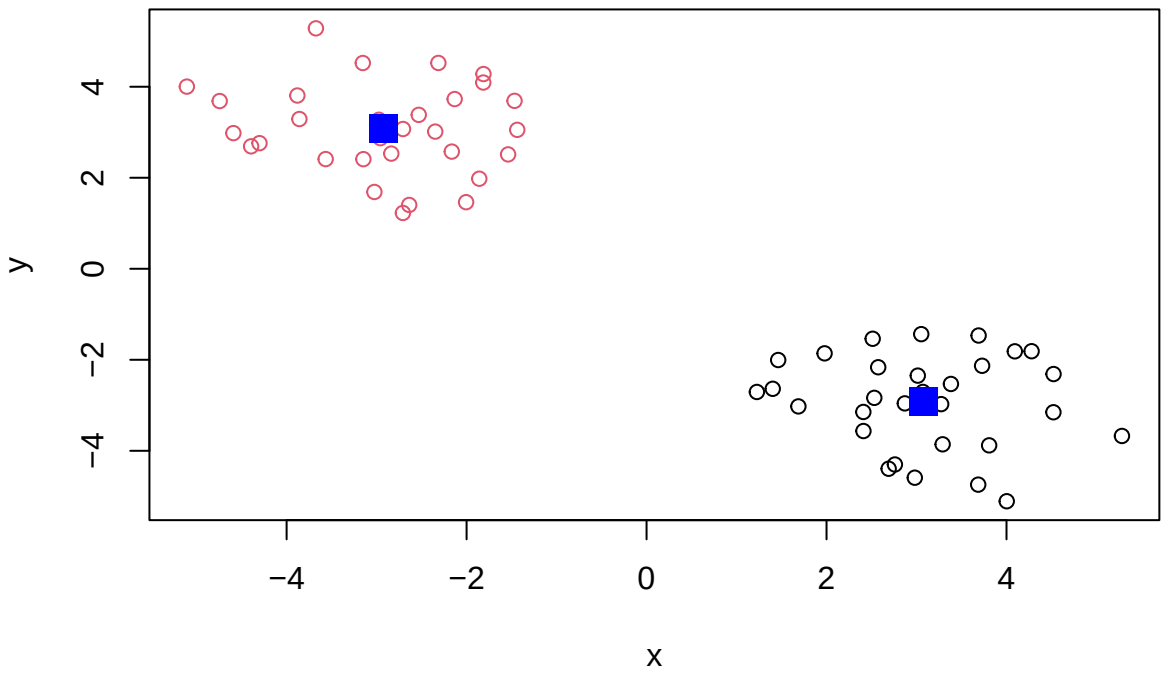
Q. What 'component' of your result object details cluster center?

```
km$centers
```

```
##           x         y
## 1  3.073150 -2.922852
## 2 -2.922852  3.073150
```

Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```
plot(x, col=km$cluster)
points(km$centers, col="blue", pch=15, cex=2)
```




#Hierarchical Clustering

A big limitation with kmeans() is that we have to tell it K (the number of clusters we want).

Analyze this data with hclust()

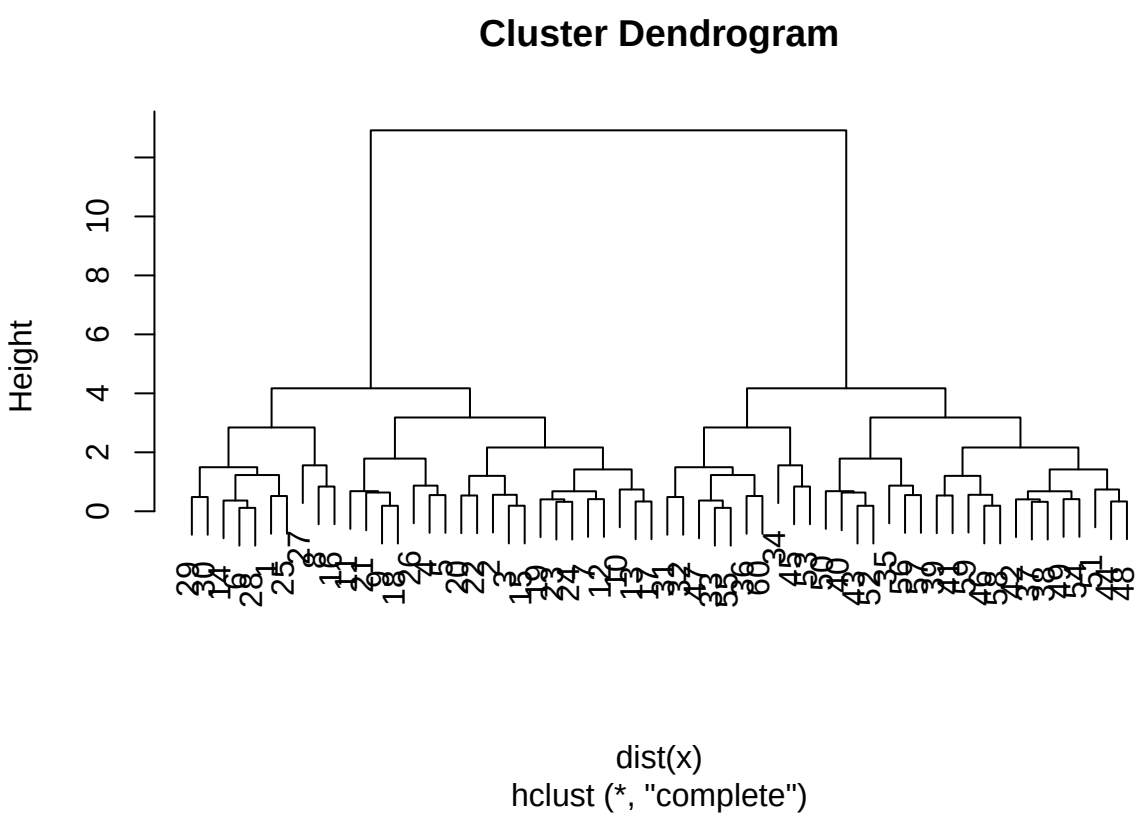
Demonstrate use of dist(), hclust(), plot() and cutree() functions to do clustern, Generate denfrograms and return cluster assignment/membership vector...

```
hc <- hclust(dist(x))
hc
```

```
##
## Call:
## hclust(d = dist(x))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

There is a plot method for hclust result objects. Let's see it

```
plot(hc)
```




To get our cluster membership vector we have to do a bit more work, we have to "cut" the tree where we think it makes sense. For this cute the 'cutree()' function

```
cutree(hc, h=6)
```

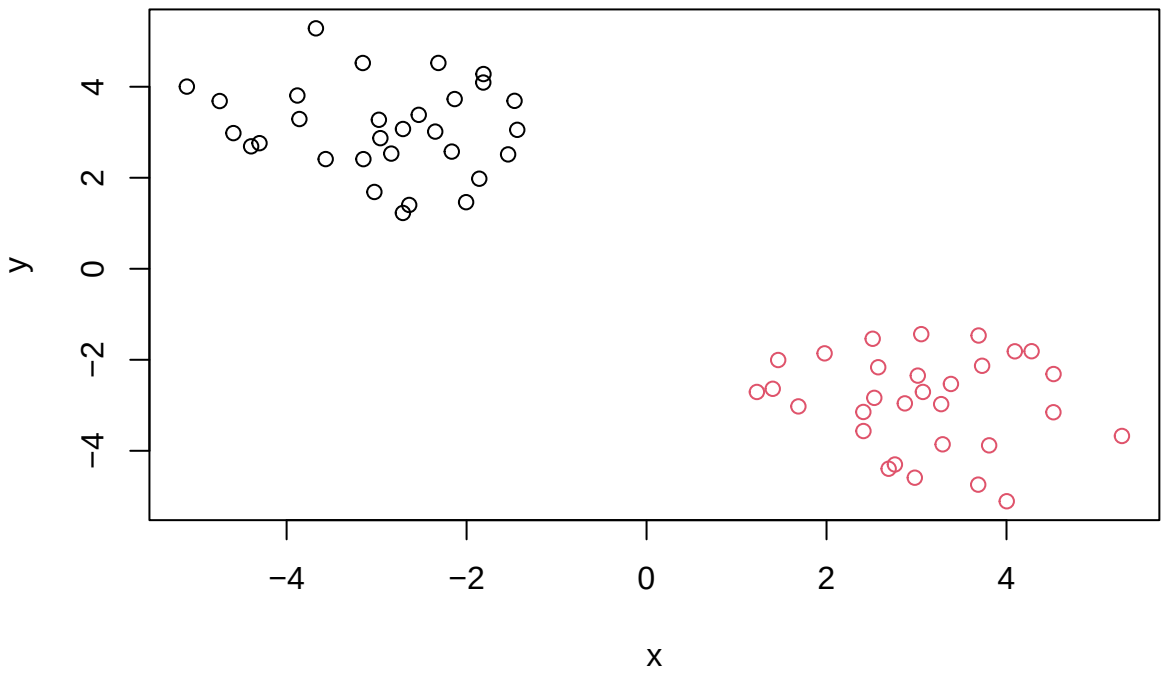
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

You can also call 'cutree' setting k = the number of grps/clusters that you want

```
grps <- cutree(hc, k=2)
```

Make our results plot

```
plot(x, col=grps)
```




Now to get started on the rest of the PCA work for this class

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
```

Q1. How many rows and colums are there in this data set?

```
ncol(x)
```

```
## [1] 5
```

```
nrow(x)
```

```
## [1] 17
```

```
dim(x)
```

```
## [1] 17  5
```

There are 5 columns and 17 rows of data in this set

```
View(x)
```

But we have an extra column! To fix this we do the following

```
rownames(x) <- x[,1]
x <- x[,-1]
head(x)
```

```
##                England Wales Scotland N.Ireland
## Cheese             105   103      103        66
## Carcass_meat       245   227      242       267
## Other_meat         685   803      750       586
## Fish               147   160      122        93
## Fats_and_oils      193   235      184       209
## Sugars             156   175      147       139
```

And it works :)) BUT if we run it again then we lose countries one by one, so there's another way to do it

```
x <- read.csv("https://tinyurl.com/UK-foods", row.names=1)
head(x)
```

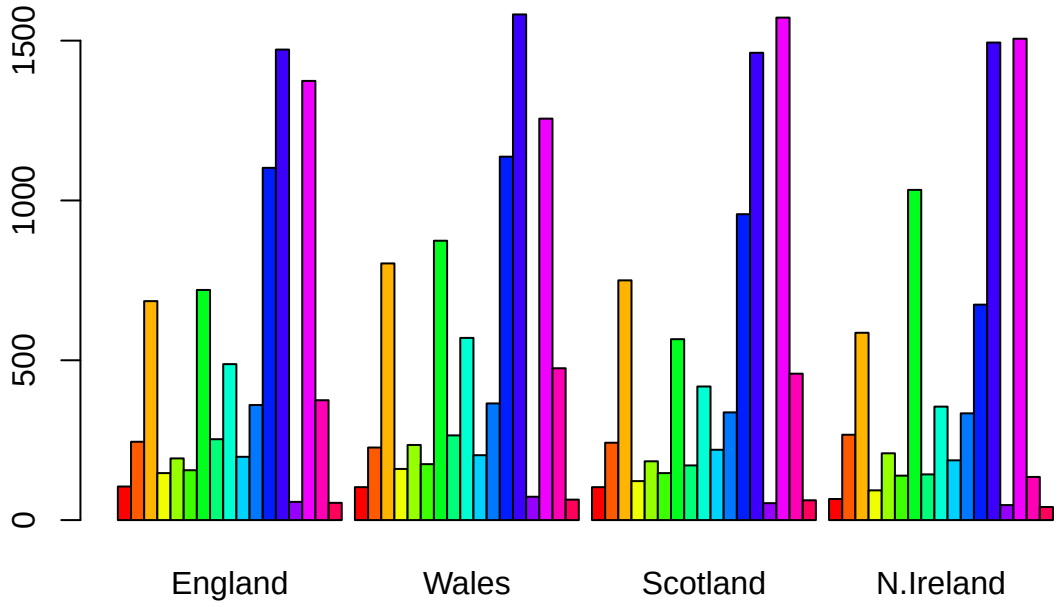
```
##                England Wales Scotland N.Ireland
## Cheese             105   103      103        66
## Carcass_meat       245   227      242       267
## Other_meat         685   803      750       586
## Fish               147   160      122        93
## Fats_and_oils      193   235      184       209
## Sugars             156   175      147       139
```

> Q2. Which approach to solving the "row-names problem' mentioned above is preferred/why? Is one approach more robust than another under certain circumstances?

I would prefer to use the second method because while both do the same thing initially, the first method could potentially delete actual data that is needed if we accidentally run it again. Meanwhile the second method, no matter how many times it is run, will always result in the end product that we desire.

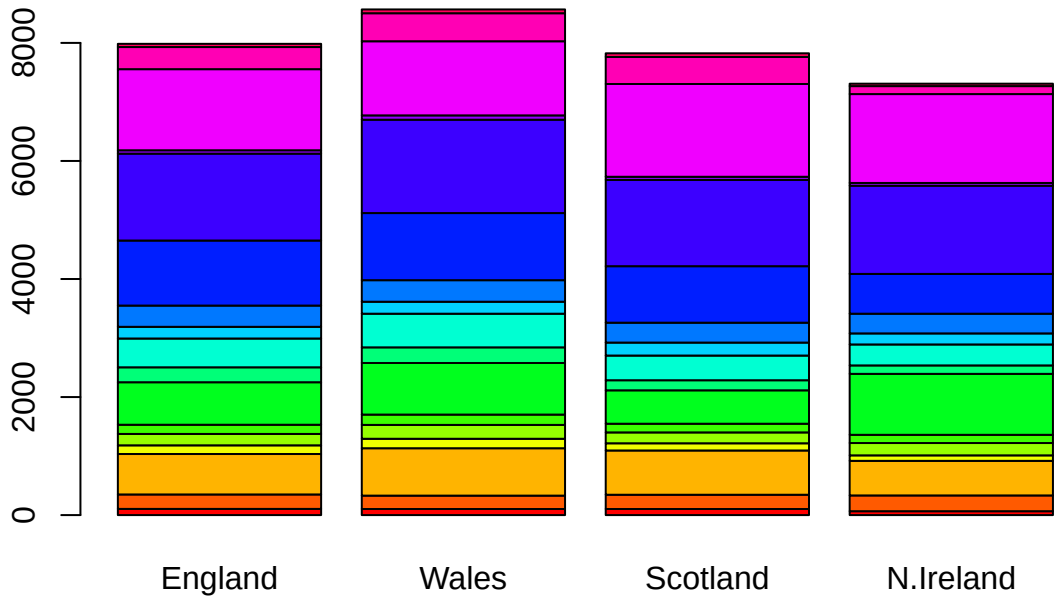
Let's try a barplot now!

```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```




Q3. What can we change in the code to change the above plot into one column for each country?

```
barplot(as.matrix(x), beside=FALSE, col=rainbow(nrow(x)))
```

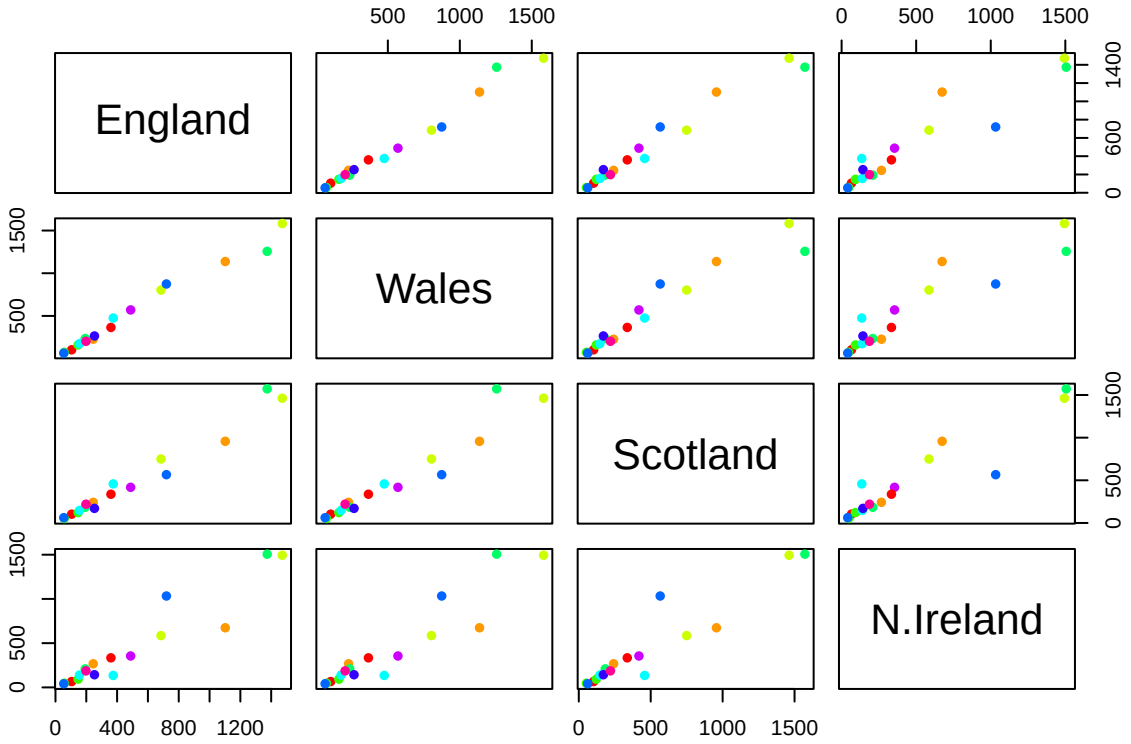



All we did was make the argument 'beside=FALSE', which caused the data to stack instead of be placed side by side.

> Q5. Generating all pairwise plots may help somewhat. Can you make sense of the following code and resulting figure? What does it mean if a given point lies on the diagonal for a given plot?

Now we will make some pairwise plots

```
pairs(x, col=rainbow(10), pch=16)
```




It plots each country against each other country. For example, in the first row the y-axis is England, and the x-axis is the corresponding countries (Wales, Scotland, N. Ireland).

So, if a point value is on the diagnol, it means that the data for both countries being compared is either the same or very similar.

> Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

To find the main differences, we look at the data points that are not on the diagnol. When comparing N. Ireland to the other countries, the dark blue point is off the diagnol when comparing against all 3 countries and the orange point is off when comparing with England and Wales (still slightly off for Scotland). N. Ireland consumes more fresh potatoes (blue dot is above the diagnol) and less fresh fruit (orange dot is below the diagnol) than the other countries.

#PCA to the resuce!

The main function in base R for PCA is 'prcomp()' This wants the transpose of our data.

```
pca <- prcomp(t(x))
summary(pca)
```

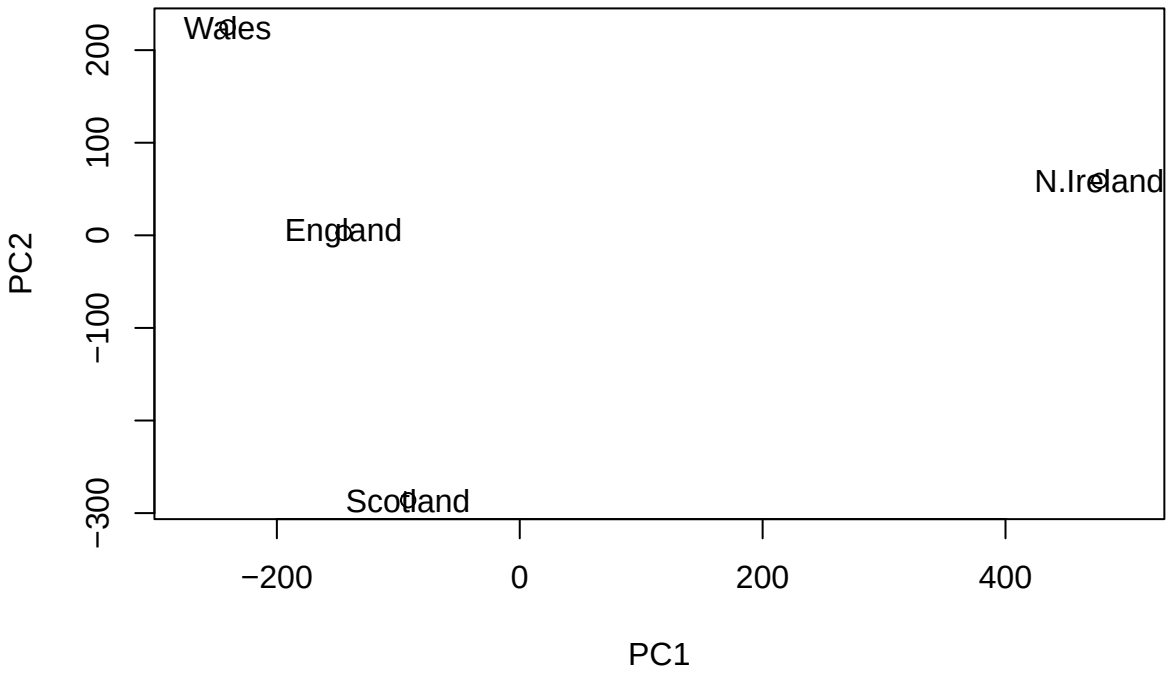
```
## Importance of components:
##                             PC1      PC2      PC3        PC4
## Standard deviation     324.1502 212.7478 73.87622 4.189e-14
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```
attributes(pca)
```

```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

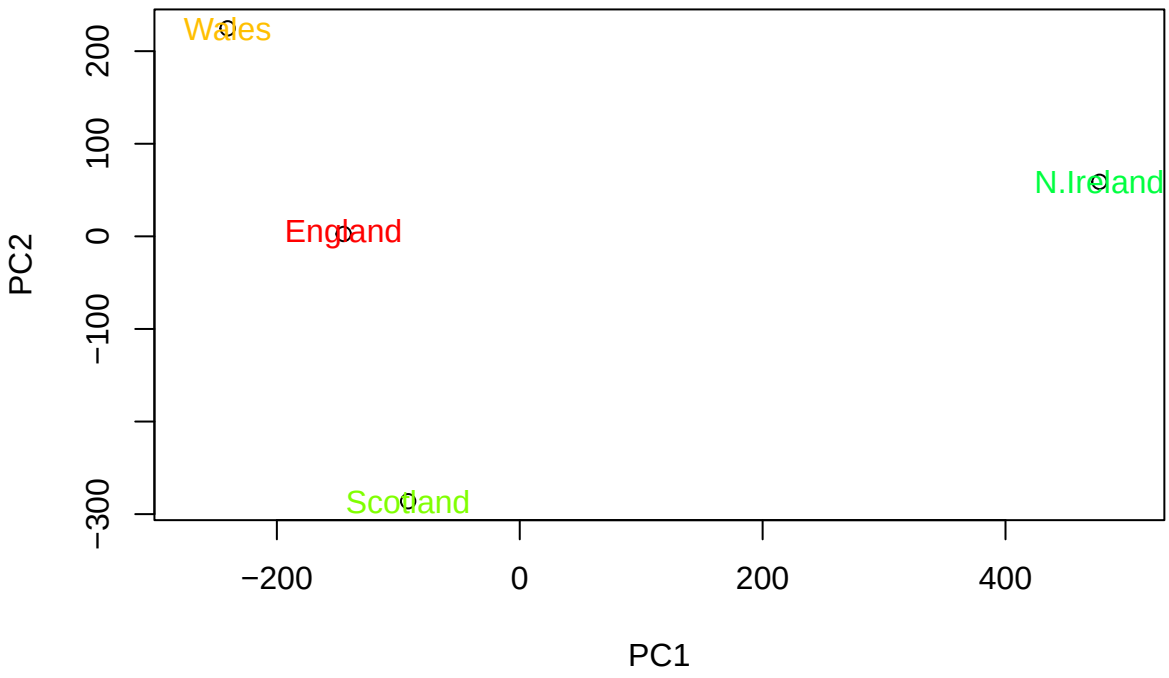
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```




Q8. Customize your plot so that the colors of the country names match the colors in our UK and Ireland map and table at start of this document.

```
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x), col=rainbow(8))
```




Calculate variation

```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

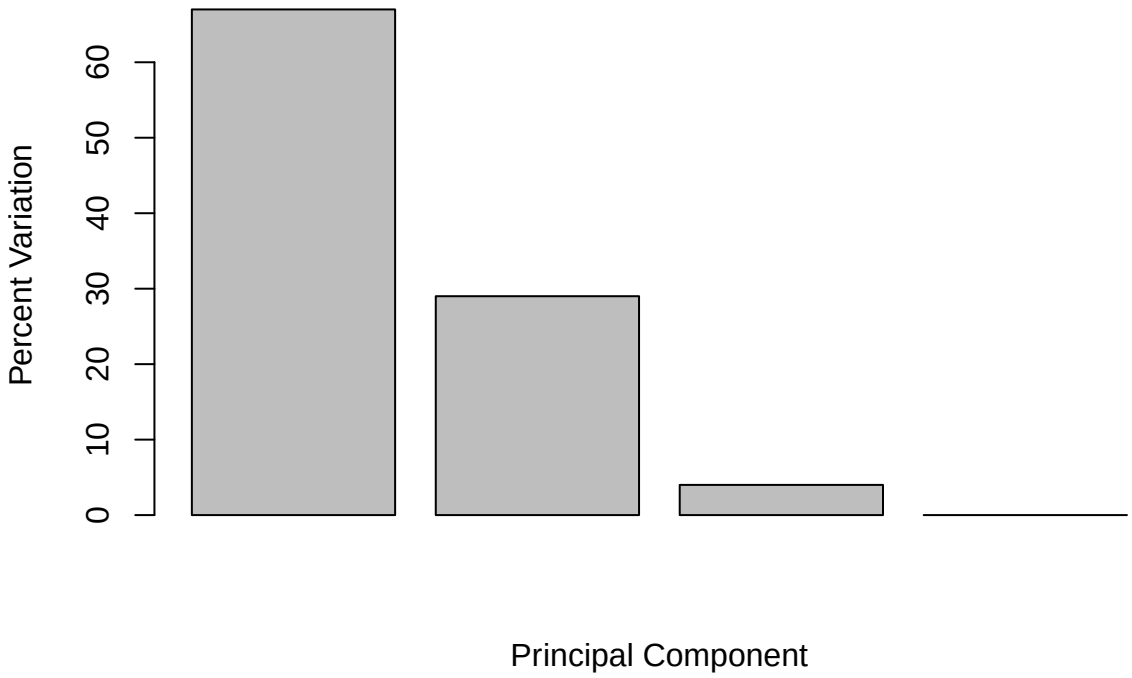
```
## [1] 67 29  4  0
```

```
## or the second row here...
z <- summary(pca)
z$importance
```

```
##                             PC1       PC2      PC3           PC4
## Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
## Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
## Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

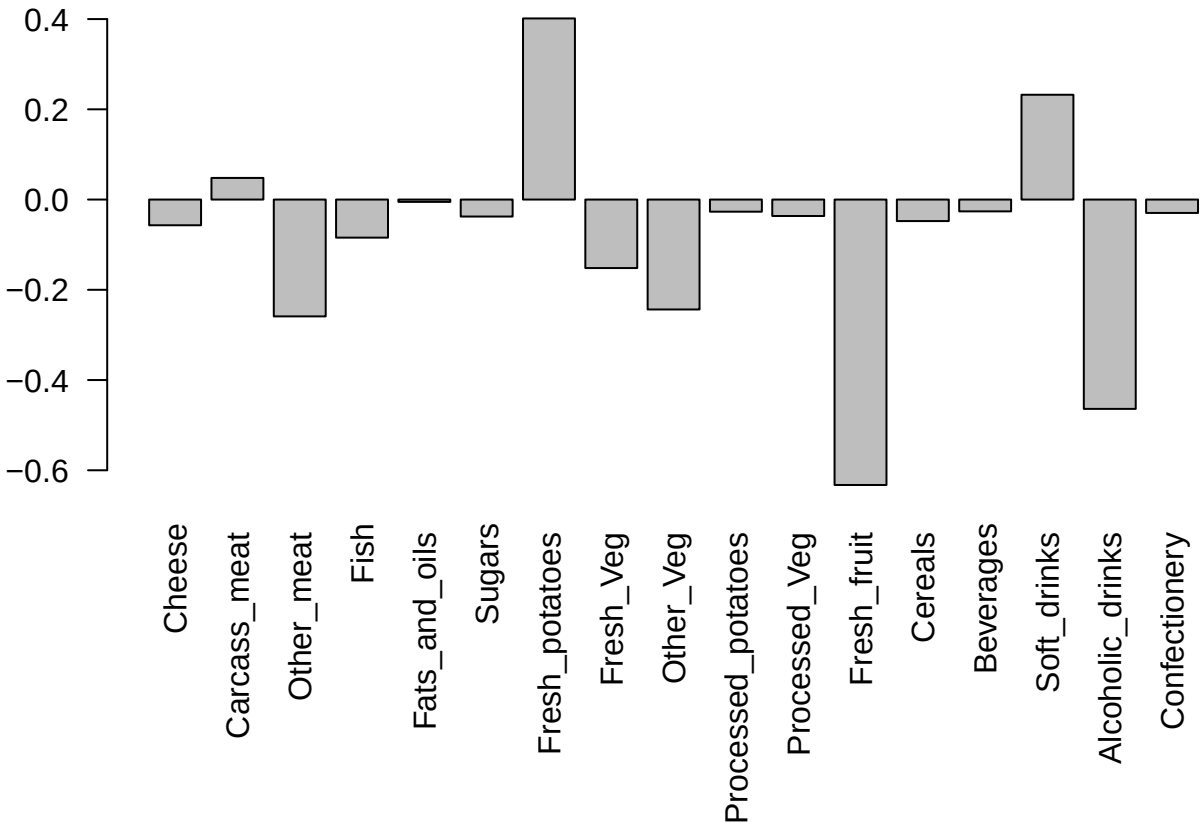
Now lets make a new barplot with this information

```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



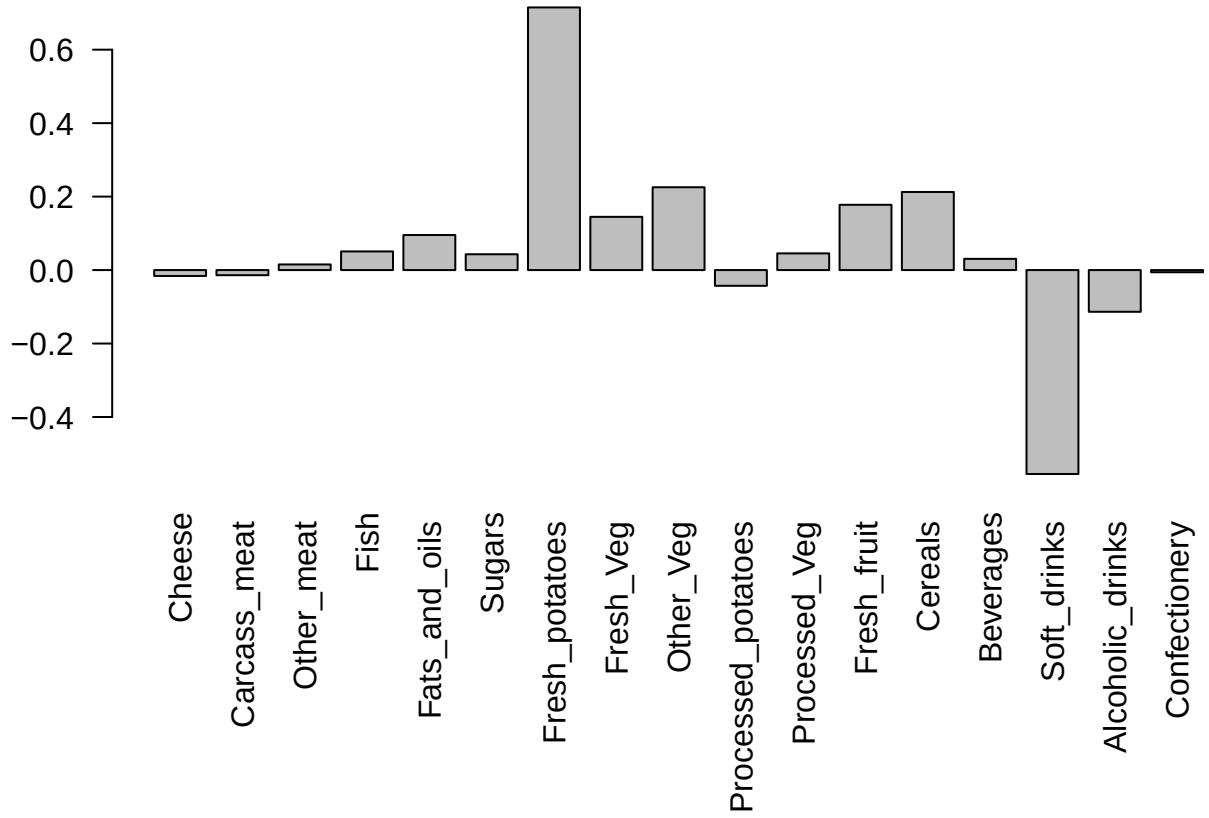

Lets focus on PC1 as it accounts for $> 90\%$ of variance

```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```




Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

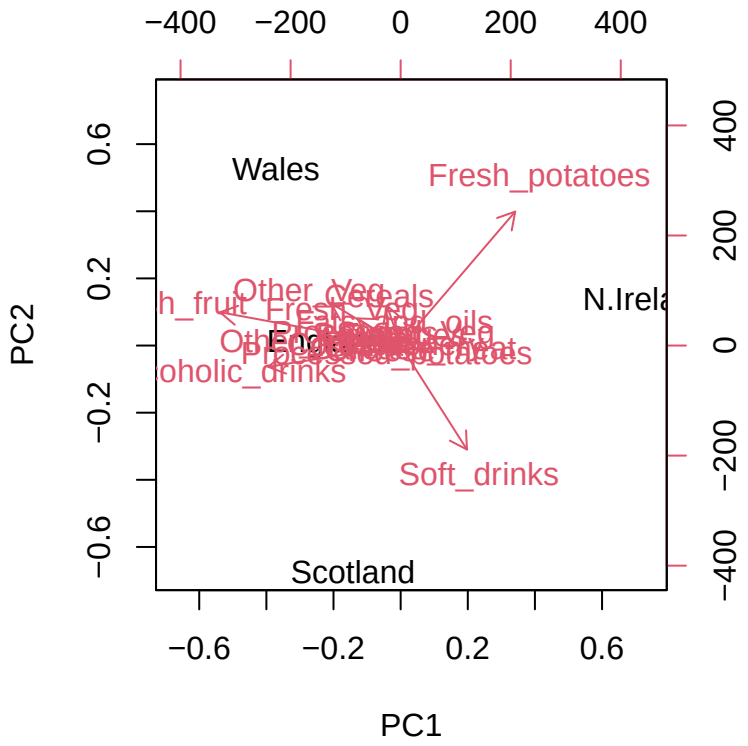
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```




Soft drinks and Fresh potatoes are most distinguishable. PC2 explains the second highest percentage of variation. In this case Fresh_potatoes and Soft_drinks were the most important for PC2.

Let's make a biplot!

```
biplot(pca)
```




#Let's try PCA with some new data!

```
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <- read.csv(url2, row.names=1)
head(rna.data)
```

```
##        wt1 wt2  wt3  wt4 wt5 ko1 ko2 ko3 ko4 ko5
## gene1  439 458  408  429 420  90  88  86  90  93
## gene2  219 200  204  210 187 427 423 434 433 426
## gene3 1006 989 1030 1017 973 252 237 238 226 210
## gene4  783 792  829  856 760 849 856 835 885 894
## gene5  181 249  204  244 225 277 305 272 270 279
## gene6  460 502  491  491 493 612 594 577 618 638
```

Q10: How many genes and samples are in this data set?

```
ngenes <- nrow(rna.data)
nsamples <- ncol(rna.data)
ngenes
```

```
## [1] 100
```

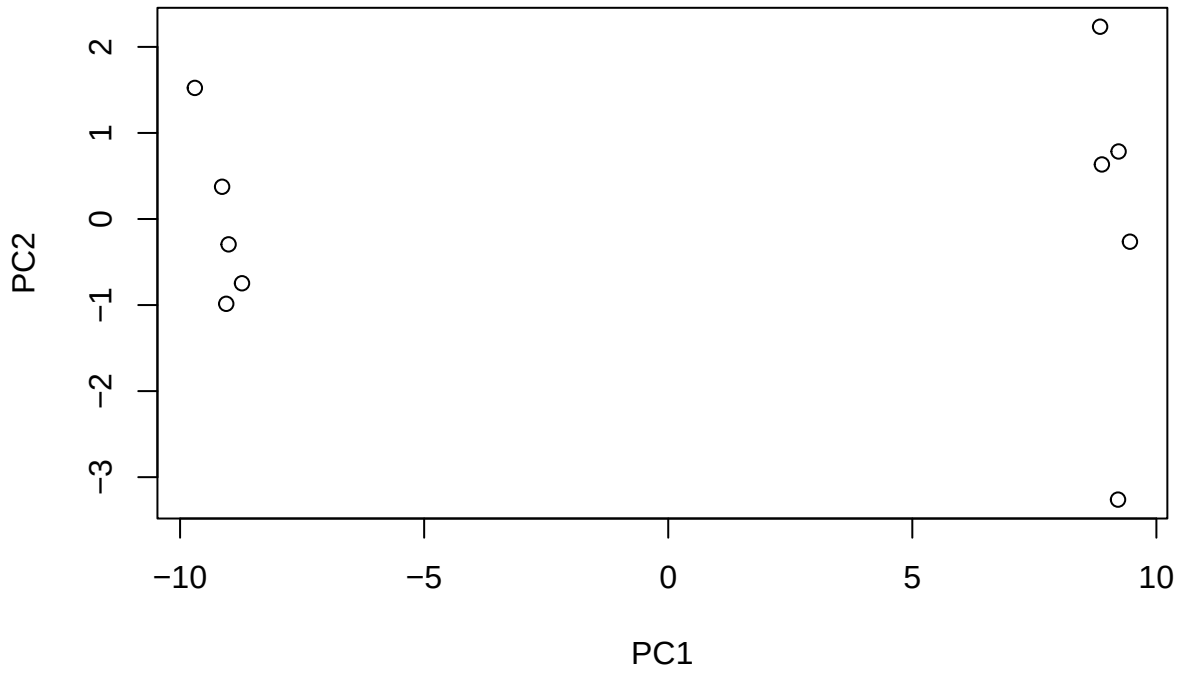
```
nsamples
```

```
## [1] 10
```

There are 100 genes and 10 samples per gene (1000 samples total)

Let's continue and use PCA to plot this data

```
pca <- prcomp(t(rna.data), scale=TRUE)

## Simple unpolished plot of pc1 and pc2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
```

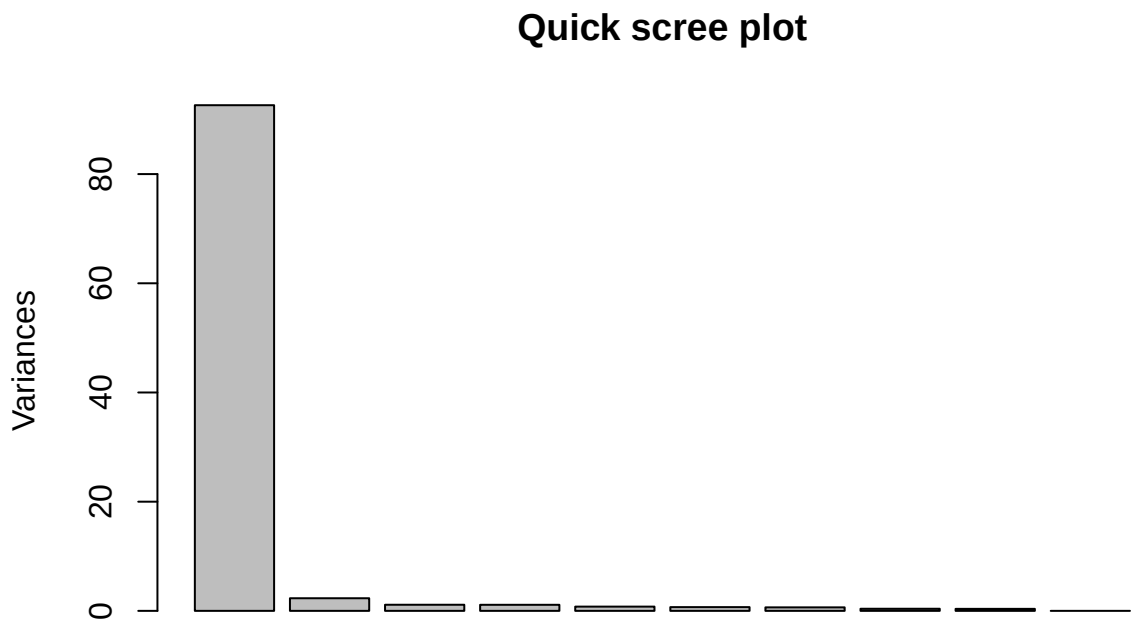



```
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.6237 1.5198 1.05787 1.05203 0.88062 0.82545 0.80111
## Proportion of Variance 0.9262 0.0231 0.01119 0.01107 0.00775 0.00681 0.00642
## Cumulative Proportion  0.9262 0.9493 0.96045 0.97152 0.97928 0.98609 0.99251
##                           PC8     PC9      PC10
## Standard deviation     0.62065 0.60342 3.348e-15
## Proportion of Variance 0.00385 0.00364 0.000e+00
## Cumulative Proportion  0.99636 1.00000 1.000e+00
```

New plot time

```
plot(pca, main="Quick scree plot")
```

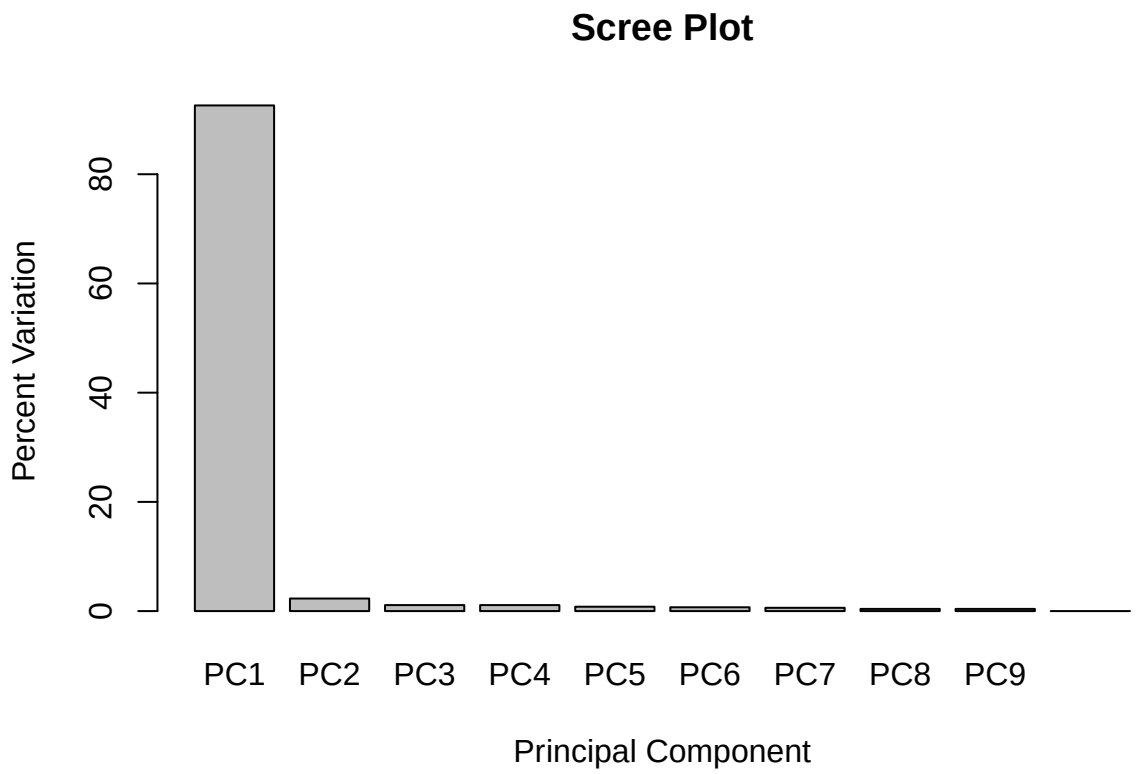



```
## Variance captured per PC
pca.var <- pca$sdev^2

## Percent variance is often more informative to look at
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
pca.var.per
```

```
##  [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

Then let's generate our own scree-plot

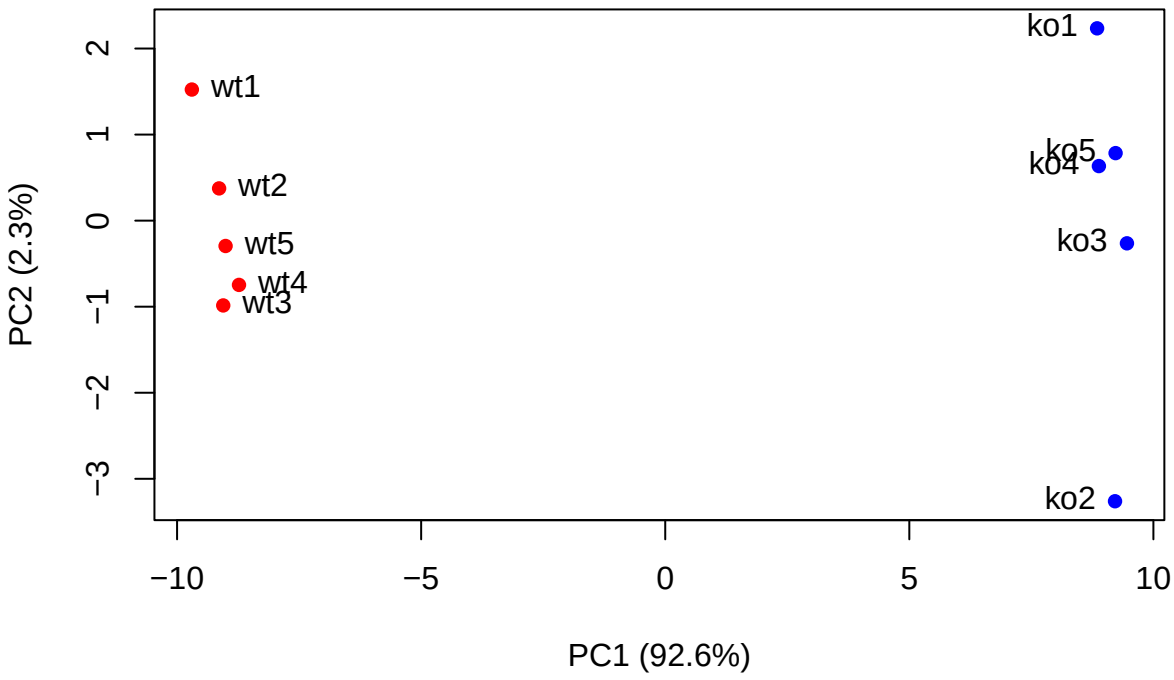
```
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```




Next we pretty it up

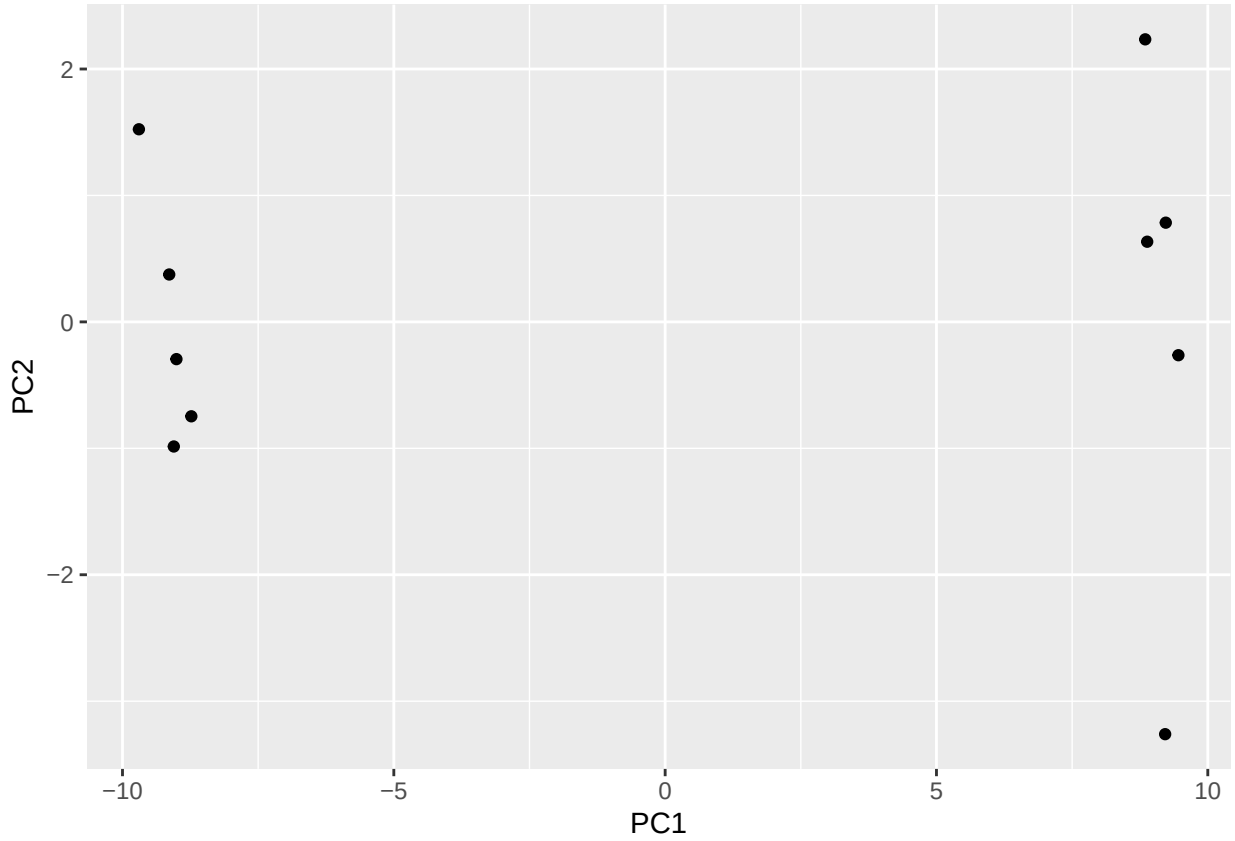
We will have a vector of colors for wt and ko samples

```
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

plot(pca$x[,1], pca$x[,2], col=colvec, pch=16,
     xlab=paste0("PC1 (", pca.var.per[1], "%)"),
     ylab=paste0("PC2 (", pca.var.per[2], "%)"))

text(pca$x[,1], pca$x[,2], labels = colnames(rna.data), pos=c(rep(4,5), rep(2,5)))
```



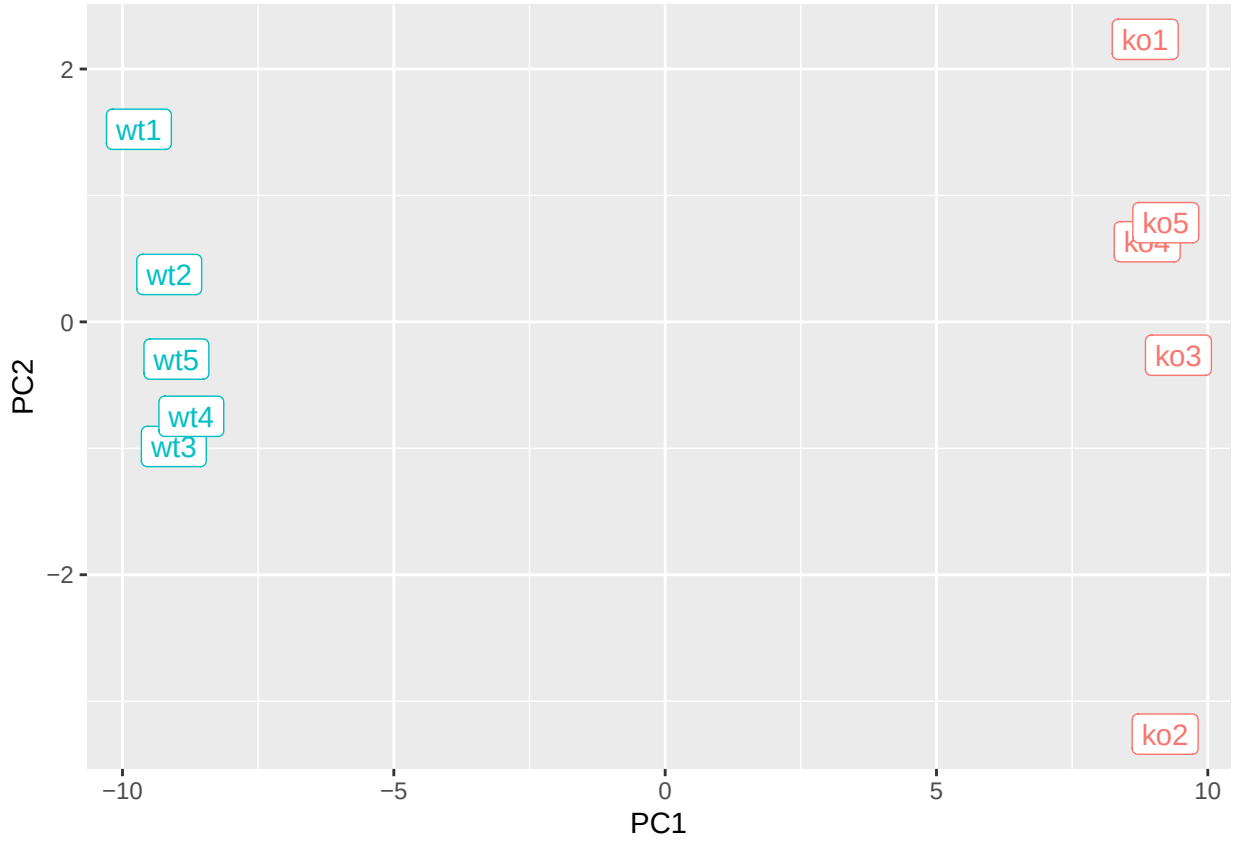

Let's go back to using ggplot!

```
library(ggplot2)
df <- as.data.frame(pca$x)

#a basic plot
ggplot(df) +
  aes(PC1, PC2) +
  geom_point()
```



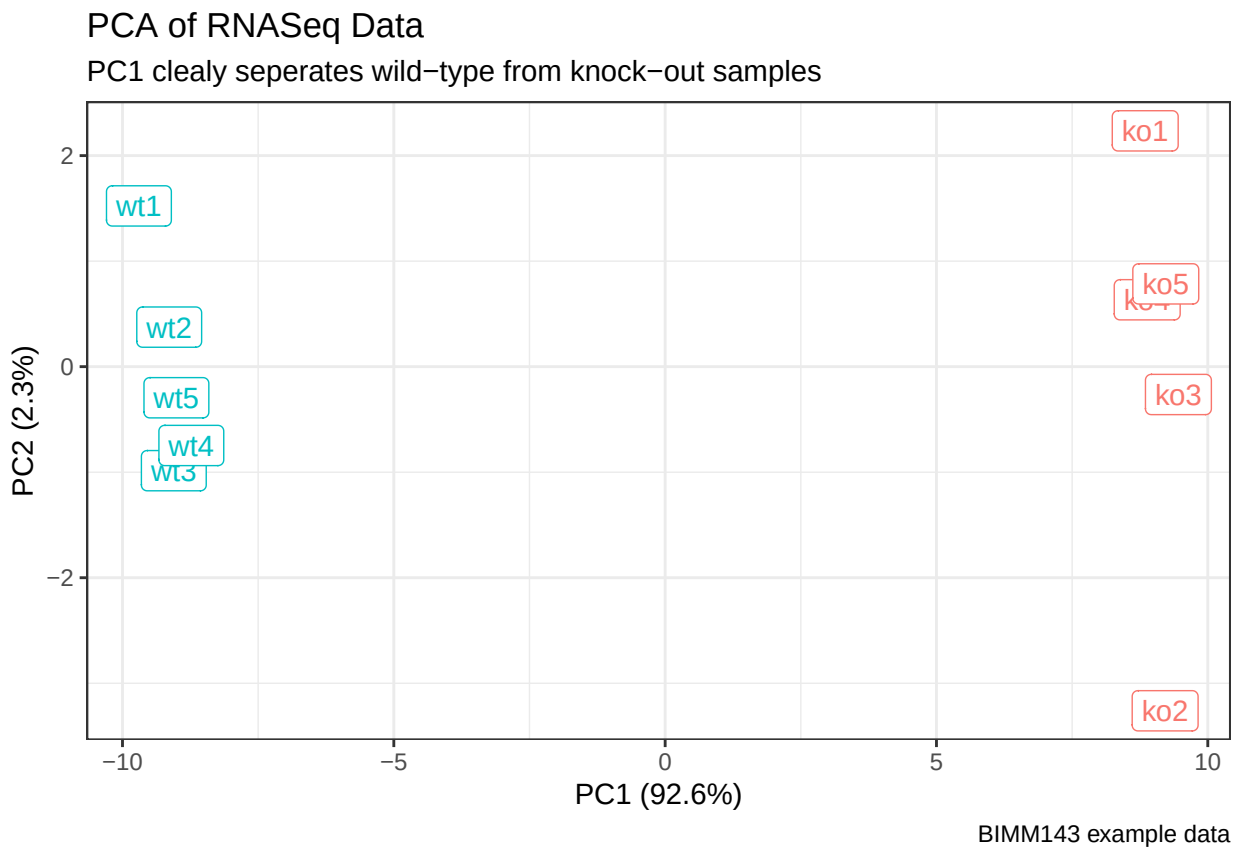

Let's make this pretty too

```
# Add a 'wt' and 'ko' "condition" column
df$samples <- colnames(rna.data)
df$condition <- substr(colnames(rna.data),1,2)

p <- ggplot(df) +
        aes(PC1, PC2, label=samples, col=condition) +
        geom_label(show.legend = FALSE)
p
```




And some more

```
p + labs(title="PCA of RNASeq Data",
       subtitle = "PC1 clealy seperates wild-type from knock-out samples",
       x=paste0("PC1 (", pca.var.per[1], "%)"),
       y=paste0("PC2 (", pca.var.per[2], "%)"),
       caption="BIMM143 example data") +
     theme_bw()
```




OPTIONAL Let's find the top 10 measurements

```
loading_scores <- pca$rotation[,1]

## Find the top 10 measurements (genes) that contribute
## most to PC1 in either direction (+ or -)
gene_scores <- abs(loading_scores)
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)

## show the names of the top 10 genes
top_10_genes <- names(gene_score_ranked[1:10])
top_10_genes
```

```
##  [1] "gene100" "gene66"  "gene45"  "gene68"  "gene98"  "gene60"  "gene21"
##  [8] "gene56"  "gene10"  "gene90"
```